卒業研究報告書

題目

MPI による wav から mp3 変換の検証

指 導 教 員 石水 隆 助教

報告者

04-1-47-200

木村惇一

近畿大学理工学部情報学科

平成 20 年 2 月 4 日提出

概要

現在、様々な分野で計算処理の高速化が求められている。高速処理を行うためには、複数のプロセッサを持つ並列計算機(Parallel Computer)が用いられる。しかし、一般に並列計算機は非常に高価であり、容易に用いることはできない。そこで、複数の計算機をネットワーク接続して 1 台の仮想的な並列計算機とする仮想並列計算(Parallel Virtual Computing)が現在重視されている。仮想並列計算を導入すれば、ネットワークを用いて誰もが安価なコストで並列計算環境を得ることができ、ベンチマーク性能ではスーパーコンピュータに匹敵する処理速度を得ることが可能となる。

本研究では、仮想並列計算環境を構築するソフトウェアの 1 つである MPI(Message Passing Interface)[1][2][3] を用いて、wav(RIFF waveform Audio Format)[9] 形式の音声データファイルを mp3(MPEG Audio Layer-3) [9][10][11]形式にエンコードしたときの実行時間を測定し、MPI の有用性を実験的に評価する。

MPI はアルゴンヌ国際研究所[4]より無料で配布されているソフトウェアであり、MPICH の公式ページ[5]からダウンロードすることにより、容易に使用することができる。

目次

第1章　序論

1.1 並列処理(Parallel Processing)と並列計算機(Parallel Computer)

並列処理(Parallel Processing)[17]は、複数のプロセッサで 1 つのタスクを動作させ、処理能力の向上を図る手法である。並列処理を用いることにより、処理時間の大幅な短縮が得られ、また、処理能力も向上すると期待されている。

並列処理を行うために用いられるのが並列計算機(Parallel Computer)[17]である。並列計算機は複数のプロセッサを持ち、各プロセッサが協調して動作することにより高い処理能力を得ることができる。

現在、並列処理が使われている例としては、並列計算機上に仮想地球を作り出し、気候変動の観測や予測などの処理を行う「地球シミュレータ」[18]などがある。

1.2 並列処理方式

並列計算の処理の方法はメモリ形式の違いから大きく分けて 2 つあり、共有メモリ(shared memory)型と分散メモリ(distributed memory)型がある。共有メモリ型並列計算機はプロセッサそれぞれが共有のメモリ空間を参照できるため、プログラミングが容易となる。しかしメモリにアクセスするネットワークが複雑化するので、プロセッサ数が多くなると並列計算機を構成するのが難しくなる。一方、分散メモリ型並列計算機は、プロセッサそれぞれがメモリを所有しており、他のプロセッサが持つデータをすぐに参照できないため、プログラミングは困難となる。しかし、多数のプロセッサを持つ並列計算機を構築することが比較的容易に可能である。このため、昨今では分散メモリ型並列計算機が主流になりつつある。

1.3 仮想並列計算(Parallel Virtual Computing)

大規模なデータの高速処理には並列計算機が必要となるが、一般に並列計算機は非常に高価であるため容易に使用することはできない。このため、複数の計算機をネットワーク接続することにより 1 台の仮想的な並列計算機とする仮想並列計算(Parallel Virtual Computing)が現在注目されている。仮想並列計算機を構築するソフトウェアの中には無償で提供されているものもあるため、安価に並列計算環境を構築することが可能である。代表的な仮想並列計算環境を構築するソフトウェアとしては、PVM(Parallel Virtual Machine)[6][7]や MPI(Message Passing Interface)[1][2][3]などが存在する。

1.4 PVM (Parallel Virtual Machine)

PVM(Parallel Virtual Machine)[6][7]は、1991 年にアメリカのオークリッジ国立研究所[8]のメンバーが中心となって開発された並列化ライブラリであり、ネットワークで接続された UNIX や NT などの複数の異機種の計算機を 1 つの並列計算機として利用することができるソフトウェアである。PVM のソフトウェア構成は、デーモンとルーチンライブラリの 2 つに分けられる[19]。

PVM を導入した計算機上で pvm3 と呼ばれるデーモンを起動すると、仮想並列計算機を構成する全ての計算機上に常駐する pvm3 デーモンを起動し、コマンドを入力することにより PVM アプリケーションを実行することができる。複数のユーザは互いに計算機をオーバラップさせて仮想並列計算機を構成でき、各ユーザは 1 人で複数の PVM アプリケーションを同時に実行させることも可能である。

PVM インタフェースを利用するためのルーチンライブラリである libpvm3.a は、メッセージパッシング、プロセスの生成、タスクの協調などの並列処理に必要な関数を提供する。

PVM は、並列計算中に仮想並列計算機を構成する計算機のうち 1 台が停止してしまった場合、停止した計算機を仮想並列計算機内から迅速に削除されるようになっているため、耐故障性に優れている。また、異機種間でも動作が可能という利点もある。しかし、PVM が多くの並列計算機に移植されるようになったとき、各並列計算機ベンダが独自にチューニングした PVM 開発を行ったため、異なるベンダ間でのプログラムの移植性が乏しくなった。

1.5 MPI (Message Passing Interface)

MPI (Message Passing Interface)[1][2][3]は並列プログラムを書くための標準ライブラリインタフェースで

ある。1990 年の初頭に、それまでベンダ独自で行っていたメッセージパッシングによる通信の仕組みを共有化することを目的に MPI の規格作成が開始され、1994 年に MPI のヴァージョン 1.0 がまとめられた。翌年 1995 年には新しい機能の拡張を考慮した MPI-2 が検討され、1997 年に規格がまとめられた。

MPI による仮想並列計算環境においての通信は TCP/IP などのネットワークを用いて行われる。仮想並列計算機を構成する各計算機はアーキテクチャにより通信方法が異なり、それに伴い実装も異なる。そのため、ユーザが通信方式の差異等を気にせずにすむように MPI では「MPI ライブラリ」が用意されている。

1.6 PVM と MPI の差異

MPI は規格を共有化することを目的としているため、PVM よりも移植性が優れている。しかし、PVM は異機種間での並列計算が可能であるのに対し、MPI は高いレベルのバッファ操作が可能であり高速にメッセージの受け渡しが可能な反面、異機種間の並列処理ができないことが欠点である。表 1 に PVM と MPI の違いを挙げる。

表 1 PVM と MPI の差異

	PVM	MPI
異機種間の並列処理	可能	不可能
アプリケーション中の動的プロセス生成・停止	可能	不可能
アプリケーション中の動的ノード数の増減・ノード数の調査	可能	不可能
移植性	低い	高い
メッセージ受け渡し	低速	高速
Windows 系 OS への対応	不充分	充分

現在では、高速なメッセージ処理および移植性の高さから、MPI の方が主流となりつつある。

1.7 本研究の目的

MPI は移植性が高く、また現在盛んに研究が行われている。そのため、本研究では仮想並列計算環境を構築するソフトウェアとして MPI を採用した。

本研究では、MPI 環境を構築するソフトウェアの 1 つである MPICH2[5]を用いて仮想並列計算環境を構築し、仮想並列計算の有用性を実験的に評価する。評価方法としては、wav(RIFF waveform Audio Format)[9]形式の音声データファイルを mp3(MPEG Audio Layer-3) [9][10][11]形式にエンコードしたときの実行時間を測定し、計算機 1 台で処理した場合と比べてどの程度時間短縮が可能となるかを検証する。

第2章　準備

2.1 使用ソフトと使用機器

本研究では、MPI を構築するソフトウェアとして MPICH2[5]を用いる。また、多くの企業や一般家庭で Windows 系の OS が用いられていることから、本研究においても Windows 系の OS を用いる。

本研究で使用する計算機の詳細を表 2 に示す。また、本研究で用いたネットワークの構成を図 1 に示す。

表 2 本研究で使用する計算機

計算機名	Kimura	Watanabe	Ishi-5	Magician
OS	Windows XP Professional	Windows XP Professional	Windows XP Professional	Windows 2000 Professional
CPU	Pentium 1000MHz	Pentium 1000MHz	Pentium4 1.9GHz	Pentium4 1.9GHz
メモリ	512MB	512MB	640MB	670MB
CPU 数	1	1	1	1

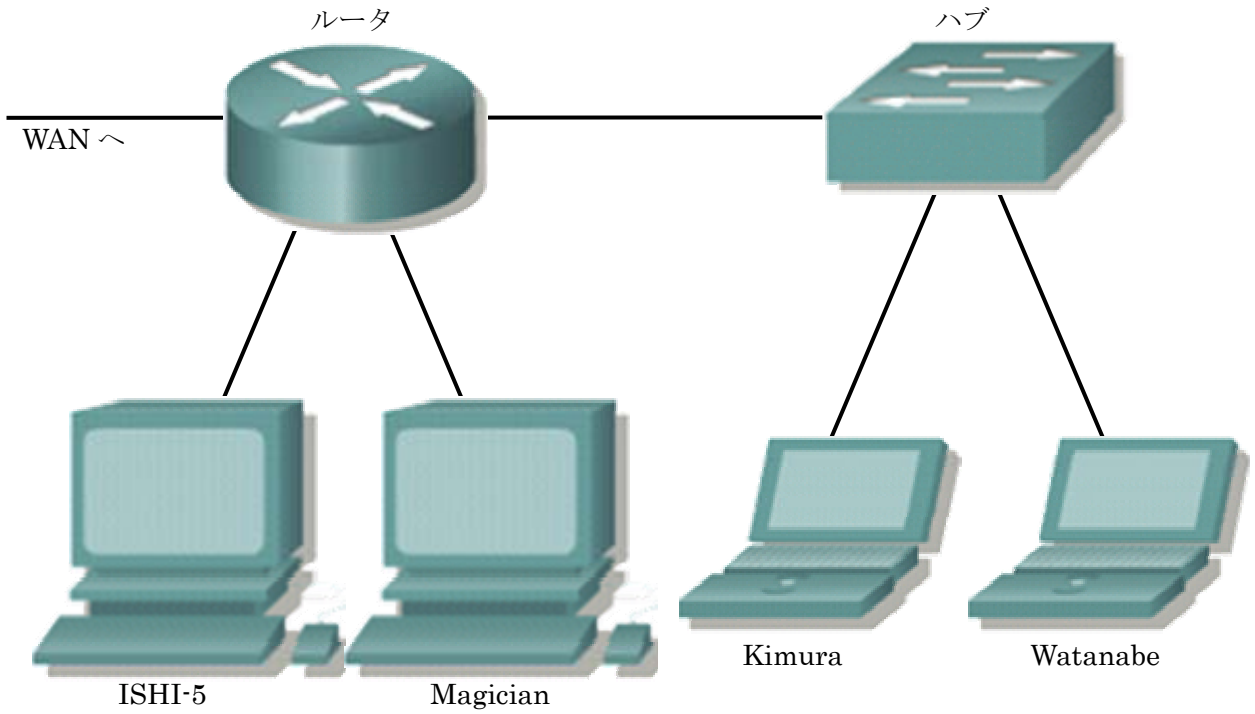

図 1 本研究で使用した計算機のネットワーク構成

2.2 MPICH-2 のインストールと環境設定

MPICH-2 を使用するために、各計算機に Windows 用の MPICH-2 のインストールを行う。MPICH-2 は、アルゴンヌ国際研究所[4]の MPICH-2 の公式ページ[5]において無償で提供されており、これをダウンロードした後各計算機にインストールする。

MPICH-2 のインストールの手順を以下に列挙する。

1. MPICH-2 の公式ページ[5]より WINDOWS 用の MPI ソフト mpich2-1.0.6p1-win32-ia32.msi をダウンロードする。
2. ダウンロードしたファイルを各計算機にインストールする。本研究では、各計算機のフォルダ”C:¥Program Files¥MPICH2”にインストールを行った。
3. MPICH-2 のバイナリのあるフォルダに対して各計算機の環境変数 PATH を指定する。本研究ではフォルダ”C:¥Program Files¥MPICH2¥bin”に対して環境変数 PATH の指定を行った。
4. 各計算機にネットワークを通して共有できるフォルダを設定する。本研究では、各計算機でフォルダ”C:¥mpi”を作成し、このフォルダのプロパティをネットワークを通じて共有できるように設定を行った。
5. 各計算機に MPICH-2 が使用するためのユーザを設定する。本研究では、各計算機に管理者権限を持つユーザ”mpi”を作成し、また、そのパスワードの設定を行った。

また、本研究では、プログラム言語として C/C++を用いた。C/C++のコンパイラは、VisualC++2005 Express Edition が、マイクロソフトの公式ページ[12]より配布されているので、その仮想 CD をダウンロードしインストールを行うことができる。

MPICH-2 はライブラリが用意されているので、VisualC++のツールオプションから MPICH-2 のインクルードファイルおよびライブラリファイルのあるフォルダ”C:¥Program Files¥MPICH2¥include”および”C:¥Program Files¥MPICH2¥lib”を追加し、リンカ入力の依存ファイル”mpi.lib”を追加することにより、MPICH-2 を用いて並列プログラムを作成する環境を作ることができる。

2.3 音声ファイル

本研究では、MPICH-2 を用いて、wav(RIFF waveform Audio Format)[9]形式の音声データファイルを mp3(MPEG Audio Layer-3) [9][10][11]形式にエンコードを行う。wav とは、Microsoft[13]と IBM[14]により開発された音声ファイルの形式である。また、mp3 とは、MPEG-1[9]で利用されている音声圧縮方式である。

表 3 に本研究で入力として用いた wav ファイルおよび MPI により作成された mp3 ファイルの詳細を示す。

表 3 入力する wav ファイルと作成された mp3 ファイル

	入力する wav ファイル	作成された mp3 ファイル
サイズ	35.9MB(37683244b)	3.25MB(3418488B)
ビットレート	1411kbps	128kbps
オーディオサンプルサイズ	16b	
チャンネル	2(ステレオ)	2(ステレオ)
オーディオサンプルレート	44kHz	44kHz
オーディオ形式	PCM	
時間	3 分 33.6 秒	3 分 33.6 秒

第3章　mp3 の並列エンコード

3.1 mp3 のエンコード

本研究では、MPI を用いて複数の wav ファイルを mp3 にエンコードし、1 台で処理した場合と比較してどの程度時間が短縮できるかを検証する。簡単のために、使用する wav ファイルは名前のみが異なる同一のファイルを用いる。

エンコードに使用する計算機数を 1～4 台で変えながら 1,2,4,8 個の wav ファイルを mp3 にエンコードするのにかかる処理時間の計測を行う。誤差を避けるために、本研究ではそれぞれの各ケース 5 回測定を行いその平均を取る。

3.2 mp3 エンコーダ

本研究で使用したエンコーダは、lame[16]と呼ばれるエンコーダを使っているダイナミックリングライブラリ gogo.dll[15]を使用した。gogo.dll は wav から mp3 へのエンコーダを簡略化して提供している。現在 gogo.dll は LPGL ライセンスによって配布されており、ソースコードのみの配布となっている。

以下に gogo.dll を用いての mp3 へのエンコード手順について説明する。

1. gogo.dll をメモリへ読み込む。
2. ワークエリアの初期関数を呼び出す。
3. エンコード条件を設定する。
4. 条件の確定関数を呼び出す。
5. (必要であれば)確定した条件を取得する。
6. 1 フレームのエンコード関数を繰り返して呼び出す。
7. エンコード終了の関数を呼び出す。
8. gogo.dll の終了処理関数を呼び出す。
9. gogo.dll の開放をする。

複数のファイルをエンコードする場合、2.～8.を繰り返し呼び出す。

3.3 並列エンコード

本章では、本研究で作成した並列エンコーダの実行手順について述べる。

まず実行前に、以下の準備を行う。

1. 本研究で作成した並列エンコーダの実行ファイルを各計算機の”C:¥mpi”フォルダに置く。
2. 入力となる wav ファイルを各計算機の”C:¥mpi”フォルダに置く。ただし、wav ファイルのファイル名はそれぞれ”audio_1.wav”,”audio_2.wav”…とする。

MPICH は、実行時に各プロセスにランクが自動的に割り当てられる。そこで、各プロセスへの wav ファイルの振り分けはこのランクにより行うことができる。つまり、ランク n のプロセスに対しては”audio_n.wav”を割り当てれば良い。ここで注意しておくことは、wav ファイルの数以上にランクを指定してエンコードすることはできないことである。例えば、エンコードする wav ファイルが 8 個しか無いのに MPICH でプロセス数を 10 にすることはできない。ランク 9,10 を持つプロセスはそれぞれ”C:¥mpi¥audio_9.wav”および”C:¥mpi¥audio_10.wav”,を開こうとするのでエラーが起きるからである。計算機台数は何台あっても問題無いが、エンコードするファイルの数をプロセス数と同じにしなければエラーが発生する。

図 2 に並列エンコードの実行の概念図を示す。図 2 の各計算機が持つランクはあくまで例であり異なる場合(計算機 2 がランク 5 を持つなど)がある。各計算機は割り当てられた wav ファイルを mp3 に変換し、実行後はそれぞれのホストに変換された mp3 ファイルが保存される。

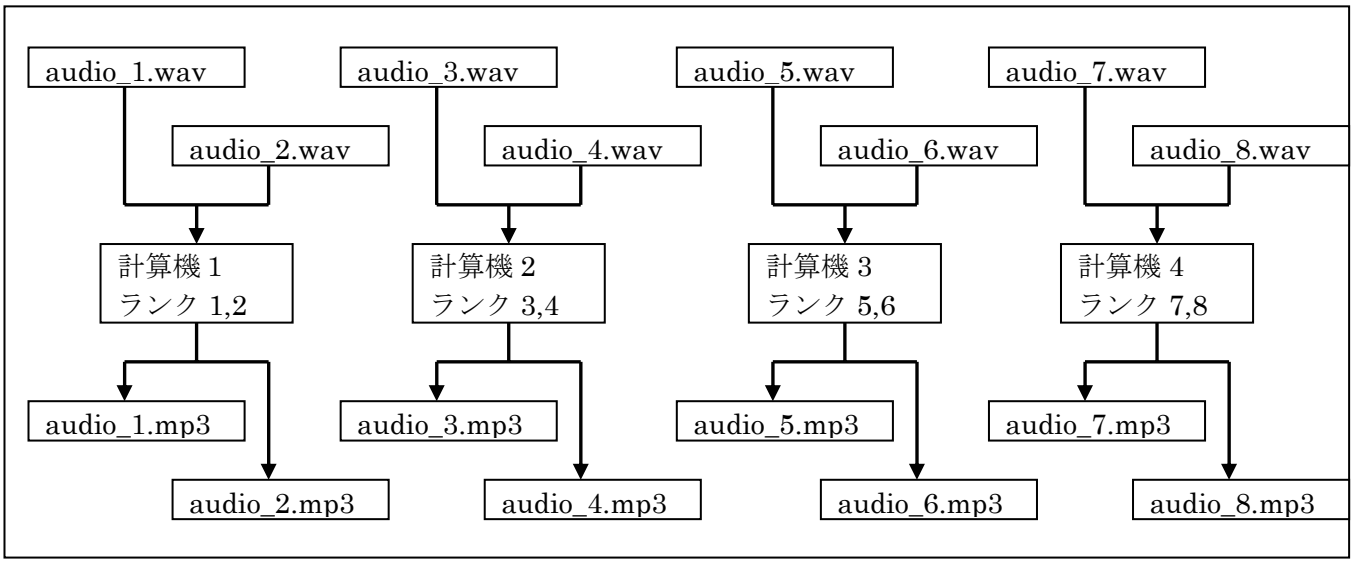

図 2 並列エンコードの実行の概念図

第4章　結果および考察

本研究では、wav ファイル数と計算機台数を変えながら mp3 に変換させ、その処理時間の計測を行った。誤差を避けるため各計測は５回行いその平均を取る。図 3 に本研究での計測結果を示す。

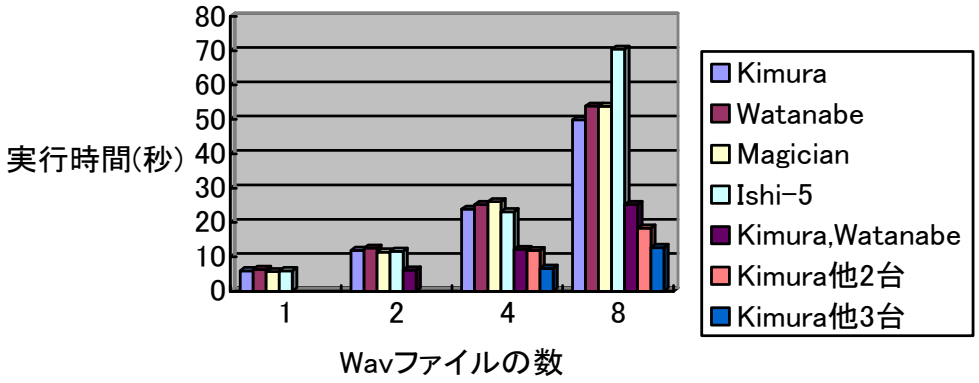

図 3 各計算機台数におけるエンコード時間

本研究において計算機台数が 2 台以上で wav ファイルが 1 つのときの計測を行わなかったのは、計算機が 2 台以上あっても処理を行うのは 1 台の計算機のみであるためである。計算機台数が 3 台以上で wav ファイルが 2 つのときも同様の理由で計測を行っていない。

図 3 より、wav ファイルの処理時間は計算機 1 台の場合に比べて計算機 4 台の場合はおよそ 1/4 となっている。従って、wav ファイルから mp3 ファイルへのエンコードでは、計算機台数の増加により、効率良く実行時間の短縮が得られたことが示される。

第5章　結論

本研究では、MPICH2[5]による仮想並列計算環境の下で wav 形式のファイルから mp3 形式のファイルへのエンコードを行い、その実行時間を測定することで、仮想並列計算の有用性を実験的に評価した。

本研究での計測結果より、wav ファイルから mp3 ファイルへのエンコードでは、計算機台数の増加により、効率良く実行時間の短縮が得られたことが示された。従って、MPI による仮想並列計算環境の構築は非常に有用であると考えられる。

しかし、本研究で作成した並列エンコーダは、仮想並列計算機を構築する各計算機の性能差を考慮していないために、極端に低スペックと高スペックの計算機が入り混じった状況では充分な速度向上が得られない可能性がある。データを各計算機に均等に割り当てるのでは無く、高スペックの計算機に多くのデータを割り当て、低スペックの計算機の負荷を減らせば、仮想並列計算機全体の性能が向上する可能性がある。従って、仮想並列計算機を構成する各計算機のスペックに応じて割り当てるデータ数を適宜変えていくアルゴリズムを開発することが今後の課題である。

謝辞

本研究を行うにあたり、並列処理の基礎から自分達の研究内容まで様々なご指導ご鞭撻をいただき、石水隆助教にはまことに感謝しております。また、同じ研究を目的とした共同研究者の阪木君、渡辺君には深い感謝、敬愛の気持ちを表します。

参考文献

[1] P.パチェコ 著, 秋葉博 訳：MPI 並列プログラミング, 培風館（2001）

[2] W.グロップ,E.ラスク,T.タークル著, 畑崎隆雄 訳：実践 MPI-2 メッセージパッシング・インターフェースの上級者向け機能, ピアソン・エデュケーション(2002)

[3] 渡邊真也 著：MPI による並列プログラミングの基礎, http://mikilab.doshisha.ac.jp/dia/smpp/cluster2000/PDF/chapter02.pdf

[4] Argonne National Laboratory, http://www.mcs.anl.gov/research/projects/mpich2/indexold.html

[5] MPICH2, http://www.mcs.anl.gov/research/projects/mpich2/

[6] PVM, Parallel Virtual Machine, http://www.csm.ornl.gov/pvm/

[7] PVM, http://erpc1.naruto-u.ac.jp/~geant4/pvm/pvm.html

[8] OAK RIDGE National Laboratory, http://ww.ornl.gov/

[9] 第一 I/O 編集部 編：音声・動画・文書ファイルの形式の達人になる本, 工学社(2002)

[10] 大沢文考：たちまちわかる MP3, 工学社(1999)

[11] ch3,恣岡悄：特集 サウンドフォーマット MP3, プログラミング技術情報誌・C マガジン, pp.26—55, Vol3 (1999)

[12] Visual Studio 2008 Express Editions, http://www.microsoft.com/japan/msdn/vstudio/express/default.aspx

[13] Microsoft, http://www.microsoft.com/ja/jp/default.aspx

[14] IBM, http://www.ibm.com/jp/

[15] 午後のこ〜だ オンラインマニュアル, http://www.marinecat.net/free/windows/gogohelp/

[16] KKKK.net, http://kkkkk.net/

[17] J.JáJá：An Introduction to Parallel Algorithms ,Addison Wesley(1992)

[18] 地球シミュレータセンター, http://www.es.jamstec.go.jp/index.html

[19] PVM, Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing, http://www.netlib.org/pvm3/book/pvm-book.html

付録 並列エンコーダプログラムのソースコード

本研究で用いた並列エンコーダのプログラムを以下に示す。

1. encoder.cpp
2. stab.cpp
3. musenc.h

また gogo.dll のソースコードを http://www.marinecat.net/free/windows/mct_free.htm からダウンロードし、コンパイルして gogo.dll を用意しておく必要がある。

付録.1. encoder.cpp

```
/*
 *  コンパイル時 stab.cpp を一緒にコンパイルしてください
 */
#define MPICH_SKIP_MPICXX
#include "mpi.h"
#include <stdio.h>
#include <windows.h>
#include "musenc.h"
#include <time.h>
#include <stdlib.h>

/*
ファイル名と拡張子を分けて後で結合する
audio_の後には任意の数字が結合され.wav が次に結合される
-->C:¥mpi¥audio_1.wav
*/
#define FILE "C:¥¥mpi¥¥audio_"

int ErrorCheck(MERET rval){
        switch(rval){
                case ME_NOERR:return 1;break;
                case ME_EMPTYSTREAM:return 1;break;
                case ME_HALTED:printf("中断されました¥n");return -1;break;
                case ME_INTERNALERROR:printf("内部エラーが発生しました¥n");return -1;break;
                case ME_PARAMERROR:printf("設定パラメーターのエラー¥n");return -1;break;
                case ME_NOFPU:printf("x87FPU を装着していません¥n");return -1;break;
                case ME_INFILE_NOFOUND:printf("入力ファイルを正しく開けません¥n");return -1;break;
                case ME_OUTFILE_NOFOUND:printf("出力ファイルを正しく開けません¥n");return -1;break;
                case ME_FREQERROR:printf("入出力周波数が正しくありません¥n");return -1;break;
                case ME_BITRATEERROR:printf("出力ビットレートが正しくありません¥n");return -1;break;
                case ME_WAVETYPE_ERR:printf("ウェーブタイプが正しくありません¥n");return -1;break;
                case ME_CANNOT_SEEK:printf("正しくシーク出来ません¥n");return -1;break;
                case ME_BITRATE_ERR:printf("ビットレート設定が正しくありません¥n");return -1;break;
                case ME_BADMODEORLAYER:printf("モードの設定が正しくありません¥n");return -1;break;
```

```
                case ME_NOMEMORY:printf("メモリアローケーションに失敗しました¥n");return -1;break;
                case ME_CANNOT_CREATE_THREAD:printf("スレッド生成エラー¥n");return -1;break;
                case ME_WRITEERROR:printf("記憶媒体の容量不足です¥n");return -1;break;
                default:return -1;
        }
}

//フレーム単位でエンコードする
MERET frame_encoder(MERET rval,UPARAM totalFrame,UPARAM curFrame)
{
        do {
                //printf("%d / %d (%d%%)¥r", curFrame,
                //              totalFrame,curFrame / ((totalFrame + 99)/100) );
                curFrame++;
                // 1 フレームエンコードを繰り返す
                rval = MPGE_processFrame();
                // 入力ストリームがなくなる(ME_EMPTYSTREAM) or
                // その他エラーが発生するまで繰り返し。
                } while(rval == ME_NOERR);

                return rval;
}

int main(int argc, char **argv)
{
        MPI_Comm mpi_comm;
        //MPI_Status mpi_stat;
        int num_proc,myrank,proc_name_len;
        char proc_name[10];

        static char filename[256];//="file" + "(myrank+1)" + "extension"
        char file[] = FILE;//audio_X.mp3 で出力する(X は任意の数字)
        char extension[]=".wav";//拡張子.wav。filename に結合するためのファイル
        MERET   rval;
        double ts,te,tp;//時間測定のため


        MPI_Init(&argc,&argv);  //MPI ライブラリを使用するための準備(初期化)を行う
        mpi_comm = MPI_COMM_WORLD;

        MPI_Comm_size(mpi_comm, &num_proc);

        MPI_Comm_rank(mpi_comm, &myrank);
        MPI_Get_processor_name(proc_name, &proc_name_len);
        MPI_Barrier(mpi_comm);
        ts=MPI_Wtime();
```

```
/****************/
//MPI 振り分け処理
/****************/
if(myrank==0){
		printf("%s is rank:%d  処理中¥n",proc_name,myrank);
		// 1. DLL 読み込み&初期化
		rval = MPGE_initializeWork();
		if(!ErrorCheck(rval))return -1;

		// 2. ファイル名の設定
		sprintf(filename,"%s%d%s",file,1,extension);

		rval=MPGE_setConfigure( MC_INPUTFILE, MC_INPDEV_FILE, (UPARAM)filename);
		if(!ErrorCheck(rval))return -1;

		// 3. パラメータ解析
		rval = MPGE_detectConfigure();
		if(!ErrorCheck(rval))return -1;

		// 全フレーム数を取得
		UPARAM   totalFrame, curFrame;
		MPGE_getConfigure( MG_COUNT_FRAME, (UPARAM*)&totalFrame);
		curFrame = 0;

		//エンコード
		rval = frame_encoder(rval,totalFrame,curFrame);

		//エンコードが終わってストリームが最後まで達したかどうか
		ErrorCheck(rval);

		// 5.エンコーダーを閉じる
		MPGE_closeCoder();
		printf("%s is rank %d:   %s -> %s%d.mp3¥n",proc_name,myrank,filename,file,myrank+1);
}
else{
		printf("%s is rank:%d  処理中¥n",proc_name,myrank);
		// 1. DLL 読み込み&初期化
		rval = MPGE_initializeWork();
		if(!ErrorCheck(rval))return -1;

		// 2. ファイル名の設定
		sprintf(filename,"%s%d%s",file,1+myrank,extension);

		rval=MPGE_setConfigure( MC_INPUTFILE, MC_INPDEV_FILE, (UPARAM)filename);
		if(!ErrorCheck(rval))return -1;

		// 3. パラメータ解析
		rval = MPGE_detectConfigure();
```

```
                if(!ErrorCheck(rval))return -1;

                // 全フレーム数を取得
                UPARAM   totalFrame, curFrame;
                MPGE_getConfigure( MG_COUNT_FRAME, (UPARAM*)&totalFrame);
                curFrame = 0;

                //エンコード
                rval = frame_encoder(rval,totalFrame,curFrame);

                //エンコードが終わってストリームが最後まで達したかどうか
                ErrorCheck(rval);

                // 5.エンコーダーを閉じる
                MPGE_closeCoder();
                printf("%s is rank %d:    %s -> %s%d.mp3¥n",proc_name,myrank,filename,file,myrank+1);
        }

        MPI_Barrier(mpi_comm);
        te=MPI_Wtime();
        tp=MPI_Wtick();

        if(myrank == 0){
                printf("Process time:%lf¥n",te-ts);
                printf("Precision:%lf¥n", tp);
        }


        // 6.DLL 終了 & 開放
        MPGE_endCoder();

        MPI_Finalize();

        return 0;
}
```

付録.2. stab.cpp

```
#include <windows.h>
#include <windowsx.h>
#include <winuser.h>
#include <stdio.h>
//#include "resource.h"
#include "musenc.h"

static      HINSTANCE           hModule = NULL;
```

```
typedef MERET (*me_init)(void);
typedef MERET (*me_setconf)(MPARAM mode, UPARAM dwPara1, UPARAM dwPara2 );
typedef MERET (*me_getconf)(MPARAM mode, void *para1 );
typedef MERET (*me_detect)();
typedef MERET (*me_procframe)();
typedef MERET (*me_close)();
typedef MERET (*me_end)();
typedef MERET (*me_getver)( unsigned long *vercode,    char *verstring );
typedef MERET (*me_haveunit)( unsigned long *unit );

static      me_init                 mpge_init;
static   me_setconf     mpge_setconf;
static      me_getconfmpge_getconf;
static      me_detect  mpge_detector;
static      me_procframe mpge_processframe;
static      me_close    mpge_close;
static      me_end                  mpge_end;
static      me_getver  mpge_getver;
static      me_haveunit mpge_haveunit;

// DLL の読み込み(最初の 1 回目のみ)とワークエリアの初期化を行います。
MERET    MPGE_initializeWork()
{
        if( hModule == NULL ){
                // (DLL が読み込まれていない場合)
                // カレントディレクトリ、及び system ディレクトリの GOGO.DLL の読み込み
                hModule = LoadLibrary("gogo.dll");
                if( hModule == NULL ){                          // DLL が見つからない場合
                        #define Key                     HKEY_CURRENT_USER
                        #define SubKey "Software¥¥MarineCat¥¥GOGO_DLL"
                        HKEY      hKey;
                        DWORD     dwType, dwKeySize;
                        LONG      lResult;
                        static    char        *szName = "INSTPATH";
                        char      szPathName[ _MAX_PATH + 8];
                        dwKeySize = sizeof( szPathName );

                        // レジストリ項目の HEY_CURENT_USER¥Software¥MarineCat¥GOGO_DLL キー以下の
                        // INSTPATH (REG_SZ)を取得します。
```

```
                if( RegOpenKeyEx(
                                Key,
                                SubKey,
                                0,
                                KEY_ALL_ACCESS,
                                &hKey ) == ERROR_SUCCESS
                ){
                        lResult = RegQueryValueEx(
                                hKey,
                                szName,
                                0,
                                &dwType,
                                (BYTE *)szPathName,
                                &dwKeySize);
                        RegCloseKey(hKey);
                        if( lResult == ERROR_SUCCESS && REG_SZ == dwType ){
                                // レジストリから取得したパスで再度 DLL の読み込みを試みる
                                hModule = LoadLibrary( szPathName );
                        }
                }
        }
        // DLL が見つからない
        if( hModule == NULL ){
//              MessageBox( "DLL の読み込みを失敗しました。¥nDLL を EXE ファイルと同じディレクトリへ複写してください¥n");
                fprintf( stderr,"DLL の読み込みを失敗しました。¥nDLL を EXE ファイルと同じディレクトリへ複写してください¥n");
                exit( -1 );
        }

        // エクスポート関数の取得
        mpge_init = (me_init )GetProcAddress( hModule, "MPGE_initializeWork" );
        mpge_setconf = (me_setconf )GetProcAddress( hModule, "MPGE_setConfigure" );
        mpge_getconf = (me_getconf )GetProcAddress( hModule, "MPGE_getConfigure" );
        mpge_detector = (me_detect )GetProcAddress( hModule, "MPGE_detectConfigure" );
        mpge_processframe = (me_procframe )GetProcAddress( hModule, "MPGE_processFrame" );
        mpge_close     = (me_close )GetProcAddress( hModule, "MPGE_closeCoder" );
        mpge_end    = (me_end )GetProcAddress( hModule, "MPGE_endCoder" );
        mpge_getver                 = (me_getver )GetProcAddress( hModule, "MPGE_getVersion" );
```

```
                mpge_haveunit= (me_haveunit )GetProcAddress( hModule, "MPGE_getUnitStates" );
        }

        // すべての関数が正常か確認する
        if( mpge_init && mpge_setconf && mpge_getconf &&
                mpge_detector && mpge_processframe && mpge_end && mpge_getver && mpge_haveunit )
                return (mpge_init)();

        // エラー
        fprintf( stderr, "DLL の内容を正しく識別することが出来ませんでした¥n");
        FreeLibrary( hModule );
        hModule = NULL;
        exit( -1 );

        return ME_NOERR;
}

MERET   MPGE_setConfigure(MPARAM mode, UPARAM dwPara1, UPARAM dwPara2 )
{
        return (mpge_setconf)( mode, dwPara1, dwPara2 );
}

MERET   MPGE_getConfigure(MPARAM mode, void *para1 )
{
        return (mpge_getconf)( mode, para1 );
}

MERET   MPGE_detectConfigure()
{
        return (mpge_detector)();
}

MERET   MPGE_processFrame()
{
        return (mpge_processframe)();
}

MERET   MPGE_closeCoder()
{
```

```
        return (mpge_close)();
}

MERET   MPGE_endCoder()
{
        MERET   val = (mpge_end)();
        if( val == ME_NOERR ){
                FreeLibrary( hModule );                 // DLL 開放
                hModule = NULL;
        }
        return val;
}

MERET   MPGE_getVersion( unsigned long *vercode,  char *verstring )
{
        return (mpge_getver)( vercode, verstring );
}

MERET   MPGE_getUnitStates( unsigned long *unit)
{
        return      (mpge_haveunit)( unit );
}
```

付録.3. musenc.h

```
#ifndef __MUSUI_H__
#define __MUSUI_H__

#include <limits.h>

typedef     signed int                          MERET;
#ifndef __os2__
typedef     unsigned long                       MPARAM;
#else
typedef   unsigned long             MUPARAM;
#endif
typedef     unsigned long                       UPARAM;

#ifdef GOGO_DLL_EXPORTS
```

```
#define		EXPORT		__declspec(dllexport)
#else
#define		EXPORT
#endif


#define	ME_NOERR		(0)		// return normally;正常終了
#define	ME_EMPTYSTREAM		(1)		// stream becomes empty;ストリームが最後に達した
#define	ME_HALTED		(2)		// stopped by user;(ユーザーの手により)中断された
#define	ME_INTERNALERROR	(10)	// internal error;  内部エラー
#define	ME_PARAMERROR		(11)	// parameters error;設定でパラメーターエラー
#define	ME_NOFPU		(12)	// no FPU;FPU を装着していない!!
#define	ME_INFILE_NOFOUND	(13)	// can't open input file;入力ファイルを正しく開けない
#define	ME_OUTFILE_NOFOUND	(14)	// can't open output file;出力ファイルを正しく開けない
#define	ME_FREQERROR		(15)	// frequency is not good;入出力周波数が正しくない
#define	ME_BITRATEERROR		(16)	// bitrate is not good;出力ビットレートが正しくない
#define	ME_WAVETYPE_ERR		(17)	// WAV format is not good;ウェーブタイプが正しくない
#define	ME_CANNOT_SEEK		(18)	// can't seek;正しくシーク出来ない
#define	ME_BITRATE_ERR		(19)	// only for compatibility;ビットレート設定が正しくない
#define	ME_BADMODEORLAYER	(20)	// mode/layer not good;モード・レイヤの設定異常
#define	ME_NOMEMORY		(21)	// fail to allocate memory;メモリアローケーション失敗
#define	ME_CANNOT_SET_SCOPE	(22)	// thread error;スレッド属性エラー(pthread only)
#define	ME_CANNOT_CREATE_THREAD	(23)	// fail to create thear;スレッド生成エラー
#define	ME_WRITEERROR		(24)	// lock of capacity of disk;記憶媒体の容量不足


// definition of call-back function for user;ユーザーのコールバック関数定義
typedef	MERET	(*MPGE_USERFUNC)(void *buf, unsigned long nLength );
#define MPGE_NULL_FUNC (MPGE_USERFUNC)NULL		// for HighC

//////////////////////////////////////////////////////////////////////////
// Configuration
//////////////////////////////////////////////////////////////////////////
// for INPUT
#define		MC_INPUTFILE		(1)
```

```
// para1 choice of input device
        #define         MC_INPDEV_FILE      (0)     // input device is file;入力デバイスはファイル
        #define         MC_INPDEV_STDIO     (1)     //          stdin;入力デバイスは標準入力
        #define         MC_INPDEV_USERFUNC  (2)     //  defined by user;入力デバイスはユーザー定義
        // para2 (必要であれば)ファイル名。ポインタを指定する
        // メモリよりエンコードの時は以下の構造体のポインタを指定する.
        struct MCP_INPDEV_USERFUNC {
            MPGE_USERFUNC   pUserFunc;      // pointer to user-function for call-back or MPGE_NULL_FUNC if none
                                            // コールバック対象のユーザー関数。未定義時 MPGE_NULL_FUNC を代入
            unsigned int    nSize;          // size of file or MC_INPDEV_MEMORY_NOSIZE if unknown
                                            // ファイルサイズ。不定の時は MC_INPDEV_MEMORY_NOSIZE を指定
            int             nBit;           // nBit = 8 or 16 ; PCM ビット深度を指定
            int             nFreq;          // input frequency ; 入力周波数の指定
            int             nChn;           // number of channel(1 or 2) ; チャネル数
        };
        #define     MC_INPDEV_MEMORY_NOSIZE     (UINT_MAX)
/*
  Using userfunction input;
  ユーザー関数利用時の挙動
  ^^^^^^^^^^^^^^^^^^^^^^^^

  ユーザーが登録した関数 UsefFunc に対して、DLL より読み込み要求が行われる。
  MERET  UserFunc_input(void *buf, unsigned long nLength );

  要求を処理する際に
    ・void *buf には nLength バイト分のデータを格納、 return ME_NOERR で抜ける
      ・ファイルの最後に達して、nLength 分読み込めない(かつ少なくとも 1 バイト以上読み込める)場合、
        memset( buf + 読み込んだデータ byte, 0, nLength - 読み込んだデータサイズ) ;
        として return ME_NOERR する。
      ・1 バイトも読めない場合は、何もせず return ME_EMPTYSTREAM; で抜ける
```

```
*/

////////////////////////////////////////////////////////////////////
// for OUTPUT ( now stdout is not support )
#define          MC_OUTPUTFILE                    (2)
// para1 choice of output device
        #define          MC_OUTDEV_FILE          (0)     // output device is file;出力デバイスはファイル
        #define          MC_OUTDEV_STDOUT        (1)     //          stdout; 出力デバイスは標準出力
        #define          MC_OUTDEV_USERFUNC      (2)     //          defined by user;出力デバイスはユーザー定義
        #define          MC_OUTDEV_USERFUNC_WITHVBRTAG (3)  //          defined by user;入力デバイスはユーザー定義/VBR タグ書き出し
// para2 pointer to file if necessary ;(必要であれば)ファイル名。ポインタ指定

/*
  Using userfunction output
  ユーザー関数利用時の挙動
  ^^^^^^^^^^^^^^^^^^^^^^^^

  ユーザーが登録した関数 UsefFunc に対して、DLL より書込み要求が行われる。
  MERET  UserFunc_output(void *buf, unsigned long nLength );

  要求を処理する際に
    ・void *buf には nLength バイト分のデータが格納されているので
        fwrite( buf, 1, nLength, fp );の様にして書き出し return ME_NOERR で抜ける.
        書き出しに失敗した時は、return ME_WRITEERROR;で抜ける.
      ・最後に buf == NULL で 1 度呼び出される. return 値は何でも良い。
    (MC_OUTDEV_USERFUNC_WITHVBRTAG で登録した際には、以下の挙動が追加される)
      ・もう一度 buf == NULL で呼び出される.この際にファイルの先頭へシークし、
        ファイル全体のサイズを return の値とする。filesize<=0 の時は終了。
        (誤って return ME_NOERR; で抜けない様に注意!! )
      ・XING-VBR タグデータが buf に、XINGVBR タグのサイズが nLength に格納されて呼び出される.
      ・最後にもう一度 buf == NULL で呼び出される.
*/

////////////////////////////////////////////////////////////////////
// mode of encoding ;エンコードタイプ
#define          MC_ENCODEMODE                    (3)
```

// para1 mode;モード設定

```
#define     MC_MODE_MONO        (0)     // mono;モノラル
#define     MC_MODE_STEREO      (1)     // stereo;ステレオ
#define     MC_MODE_JOINT       (2)     // joint-stereo;ジョイント
#define     MC_MODE_MSSTEREO    (3)     // mid/side stereo;ミッドサイド
#define     MC_MODE_DUALCHANNEL (4)     // dual channel;デュアルチャネル
```

//

// bitrate;ビットレート設定

```
#define     MC_BITRATE          (4)
```

// para1 bitrate;ビットレート 即値指定

//

// frequency of input file (force);入力で用いるサンプル周波数の強制指定

```
#define     MC_INPFREQ          (5)
```

// para1 frequency;入出力で用いるデータ

//

// frequency of output mp3 (force);出力で用いるサンプル周波数の強制指定

```
#define     MC_OUTFREQ          (6)
```

// para1 frequency;入出力で用いるデータ

//

// size ofheader if you ignore WAV-header (for example cda);エンコード開始位置の強制指定(ヘッダを無視する時)

```
#define     MC_STARTOFFSET      (7)
```

//

// psycho-acoustics ON/OFF;心理解析 ON/OFF

```
#define     MC_USEPSY           (8)
```

// PARA1 boolean(TRUE/FALSE)

//

// 16kHz low-pass filter ON/OFF;16KHz 低帯域フィルタ ON/OFF

```
#define     MC_USELPF16         (9)
```

// PARA1 boolean(TRUE/FALSE)

//

// use special UNIT, para1:boolean; ユニット指定 para1:BOOL 値

```
#define         MC_USEMMX                       (10)    // MMX
#define         MC_USE3DNOW                     (11)    // 3DNow!
#define         MC_USEKNI                       (12)    // SSE(KNI)
#define         MC_USEE3DNOW            (13)    // Enhanced 3D Now!
#define         MC_USESPC1                      (14)    // special switch for debug
#define         MC_USESPC2                      (15)    // special switch for debug

//////////////////////////////////////////////////////////////////////
// addition of TAG; ファイルタグ情報付加
#define         MC_ADDTAG                       (16)
// dwPara1    length of TAG;タグ長
// dwPara2    pointer to TAG;タグデータのポインタ

//////////////////////////////////////////////////////////////////////
// emphasis;エンファシスタイプの設定
#define         MC_EMPHASIS                     (17)
// para1 type of emphasis;エンファシスタイプの設定
        #define         MC_EMP_NONE             (0)             //  no  empahsis;エンファシスなし
(dflt)
        #define         MC_EMP_5015MS   (1)             // 50/15ms      ;エンファシス 50/15ms
        #define         MC_EMP_CCITT    (3)             // CCITT        ;エンファシス CCITT

//////////////////////////////////////////////////////////////////////
// use VBR;VBR タイプの設定
#define         MC_VBR                          (18)

//////////////////////////////////////////////////////////////////////
// SMP support para1: interger
#define         MC_CPU                          (19)

//////////////////////////////////////////////////////////////////////
// for RAW-PCM; 以下 4 つは RAW-PCM の設定のため
// byte swapping for 16bitPCM; PCM 入力時の low, high bit  変換
#define         MC_BYTE_SWAP            (20)

//////////////////////////////////////////////////////////////////////
// for 8bit PCM
#define         MC_8BIT_PCM                     (21)
```

```
//////////////////////////////////////////////////////////////////////
// for mono PCM
#define                 MC_MONO_PCM                                     (22)

//////////////////////////////////////////////////////////////////////
// for Towns SND
#define                 MC_TOWNS_SND                    (23)

//////////////////////////////////////////////////////////////////////
// BeOS & Win32 Encode thread priority
#define                 MC_THREAD_PRIORITY              (24)
// (WIN32) dwPara1 MULTITHREAD Priority (THREAD_PRIORITY_**** at WinBASE.h )

//////////////////////////////////////////////////////////////////////
// BeOS Read thread priority
//#if       defined(USE_BTHREAD)
#define                 MC_READTHREAD_PRIORITY  (25)
//#endif

//////////////////////////////////////////////////////////////////////
// output format
#define                 MC_OUTPUT_FORMAT                (26)
// para1
        #define             MC_OUTPUT_NORMAL        (0)         // mp3+TAG(see MC_ADDTAG)
        #define             MC_OUTPUT_RIFF_WAVE     (1)         // RIFF/WAVE
        #define             MC_OUTPUT_RIFF_RMP      (2)         // RIFF/RMP

//////////////////////////////////////////////////////////////////////
// LIST/INFO chunk of RIFF/WAVE or RIFF/RMP
#define                 MC_RIFF_INFO                    (27)
// para1 size of info(include info name)
// para2 pointer to info
//    offset            contents
//    0..3              info name
//    4..size of info-1 info

//////////////////////////////////////////////////////////////////////
// verify and overwrite
#define                 MC_VERIFY                       (28)
```

```
//////////////////////////////////////////////////////////////////////
// output directory
#define                 MC_OUTPUTDIR                    (29)

//////////////////////////////////////////////////////////////////////
// VBR の最低/最高ビットレートの設定
#define                 MC_VBRBITRATE                   (30)
// para1  最低ビットレート (kbps)
// para2  最高ビットレート (kbps)



//////////////////////////////////////////////////////////////////////
//  拡張フィルタの使用  LPF1, LPF2
#define                 MC_ENHANCEDFILTER               (31)
// para1 LPF1 (0-100)
// para2 LPF2 (0-100)



//////////////////////////////////////////////////////////////////////
// Joint-stereo における、ステレオ/MS ステレオの切り替えの閾値
#define                 MC_MSTHRESHOLD                  (32)
// para1 threshold   (0-100)
// para2 mspower     (0-100)

//////////////////////////////////////////////////////////////////////
// Language
#define                 MC_LANG                         (33)
// t_lang defined in message.h

MERET    EXPORT   MPGE_initializeWork();
#ifndef __os2__
MERET    EXPORT   MPGE_setConfigure(MPARAM mode, UPARAM dwPara1, UPARAM dwPara2 );
MERET    EXPORT   MPGE_getConfigure(MPARAM mode, void *para1 );
#else
MERET EXPORT    MPGE_setConfigure(MUPARAM mode, UPARAM dwPara1, UPARAM dwPara2 );
MERET EXPORT    MPGE_getConfigure(MUPARAM mode, void *para1 );
#endif
MERET    EXPORT   MPGE_detectConfigure();
```

```
#ifdef USE_BETHREAD
MERET    EXPORT   MPGE_processFrame(int *frameNum);
#else
MERET    EXPORT   MPGE_processFrame();
#endif
MERET    EXPORT   MPGE_closeCoder();
MERET    EXPORT   MPGE_endCoder();
MERET    EXPORT   MPGE_getUnitStates( unsigned long *unit );
MERET    EXPORT   MPGE_processTrack(int *frameNum);

// This function is effective for gogo.dll;このファンクションは DLL バージョンのみ有効
MERET    EXPORT   MPGE_getVersion( unsigned long *vercode,   char *verstring );
// vercode = 0x125 ->   version 1.25
// verstring          ->   "ver 1.25 1999/09/25" (allocate abobe 260bytes buffer)



////////////////////////////////////////////////////////////////////////
// for getting configuration
////////////////////////////////////////////////////////////////////////

#define          MG_INPUTFILE          (1)          // name of input file ;入力ファイル名取得
#define          MG_OUTPUTFILE         (2)          // name of output file;出力ファイル名取得
#define          MG_ENCODEMODE         (3)          // type of encoding    ;エンコードモード
#define          MG_BITRATE                (4)          // bitrate             ;ビットレート
#define          MG_INPFREQ                (5)          // input frequency     ;入力周波数
#define          MG_OUTFREQ                (6)          // output frequency    ;出力周波数
#define          MG_STARTOFFSET        (7)          // offset of input PCM;スタートオフセット
#define          MG_USEPSY                 (8)          //  psycho-acoustics    ;心理解析を使
用する/しない
#define          MG_USEMMX                 (9)          // MMX
#define          MG_USE3DNOW               (10)         // 3DNow!
#define          MG_USEKNI                 (11)         // SSE(KNI)
#define          MG_USEE3DNOW          (12)         // Enhanced 3DNow!

#define          MG_USESPC1                (13)         // special switch for debug
#define          MG_USESPC2                (14)         // special switch for debug
#define          MG_COUNT_FRAME        (15)         // amount of frame
#define          MG_NUM_OF_SAMPLES     (16)         // number of sample for 1 frame;1 フレームあたりのサンプル
```

数

```
#define		MG_MPEG_VERSION				(17)		// MPEG VERSION
#define		MG_READTHREAD_PRIORITY	(18)		// thread priority to read for BeOS


#endif /* __MUSUI_H__ */
```